**Documentatie Tema 2**

1.Obiectivul temei

Aceasta tema are ca si obiectiv crearea unui program care este capabil sa proceseze o lista de n clienti astfel incat acestia sa fie distribuiti unor m cozi. Fiecare client este caracterizat printr-un id, timp de sosire(arrival time) si timp de procesare(service time). Scopul acestui program este de a scrie intr-un fisier text la fiecare unitate de timp (timpul simularii) starea listei de clienti initiale si starea cozilor. Pentru fiecare coada este creat un nou thread. Simularea va lua sfarsit atunci cand toti clientii vor trece prin cozi si toate cozile vor fi inchise sau cand timpul curent va atinge timpul maxim de simulare.

2. Analiza problemei

2.1 Cazuri de utilizare

***Pasi de utilizare:***

1. Se deschide Command Prompt-ul in locatia fisierului.jar
2. Pentru a rula se introduc urmatoarele: “java –jar fisier.jar argument1 argument2” unde in loc de fisier se va scrie numele fisierului.jar, in cazul meu „test1.jar”, iar ca argumente va primi doua fisiere:primul fisier este cel din care programul va citi datele necesare, iar al doilea fisier va reprezenta fisierul de output unde se vor scrie datele de iesire.
3. Pentru ca primul fisier sa fie valid va trebui sa contina 6 numere intregi separate prin spatiu sau randuri noi. Primul numar va corespunde numarului de clienti, al doilea numar va reprezenta numarul de cozi, al treilea va fi timpul maxim de simulare.Urmatoarele doua vor fi timpul de sosire minim, repectiv maxim, interval in care clientii generati random vor lua valori. Ultimele doua valori reprezinta tot un interval( timpul minim de procesare si timpul maxim de procesare). La fel ca in cazul timpilor de sosire, clientii generati vor lua valori intre aceste doua limite.
4. Dupa ce s-a verificat ca argumentele corespund cerintelor de mai sus, se ruleaza programul.
5. Se asteapta in fereastra „Command Prompt” pana cand va aparea mesajul „S-a terminat de scris in fisier!”.In functie de numarul de clienti, dar mai important in functie de timpul maxim de simulare, programul va scrie mai repede sau mai lent.Deoarece am „adormit” thread-urile din clasa Manger si clasa Queue, un moment de timp se va afisa pe secunda.
6. Se verifica fisierul pe care l-am dat ca argument daca corespunde cerintelor problemei.

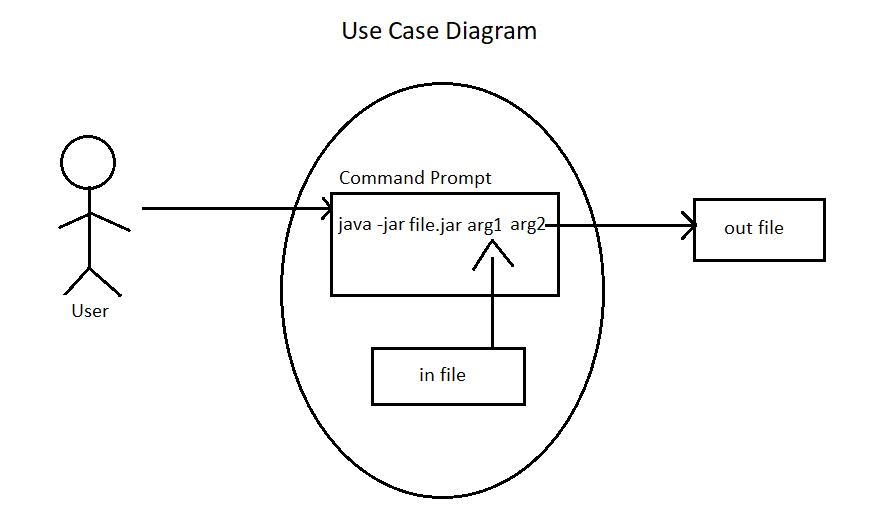
***Erori posibile:***

* Numele fisierului .jar este scris incorect.
* Fisierul de intrare nu corspunde cerintelor problemei/nu exista fisier/numele scris gresit.

***Observatii:***

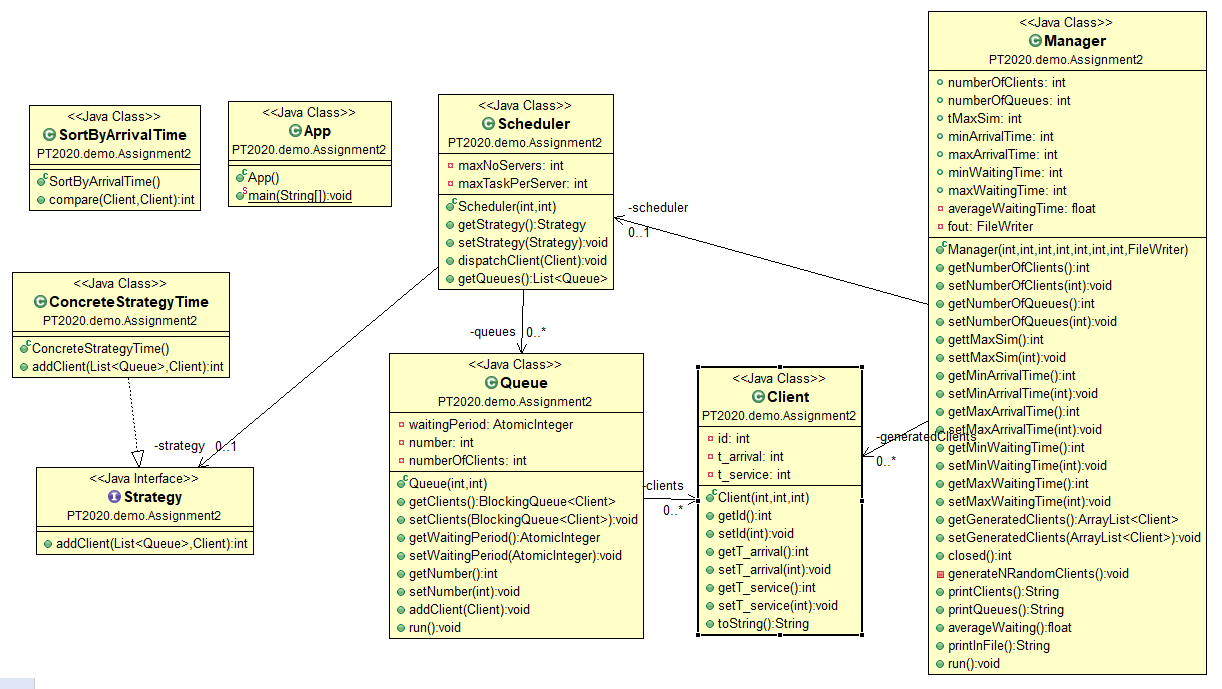
* Nu conteaza daca exista sau nu un fisier de iesire (acesta se va crea automat).
* In cazul in care exista deja un fisier de iesire, programul va suprascrie tot ce era scris in fisierul respectiv (asta in cazul in care este scris ceva).

2.2 Diagrama Use-case



3.Proiectare

3.1 Diagrama UML



3.2 Explicatii clase si algoritm utilizat

* App- este clasa in care avem metoda main. In aceasta clasa citim din fisier, cream un obiect Manager care are toate informatiile din fisier si pornim un thread pe obiectul respectiv. Fisierul din care citim este args[0], iar fisierul in care scriem este args[1].
* Clasa Manager- in constructorul clasei se primesc valorile din fisier pentru fiecare variabila instanta, cream un obiect nou de tip Scheduler in a carui constructor se vor initializa cozile de care ne vom folosi in metoda run() si generam un numar de numberOfClients clienti cu ajutorul metodei generateNRandomClients(). Dupa ce s-au generat clientii, se intra in metoda run(). Cat timp exista clienti in asteptare sau cozile nu sunt goale si timpul curent este mai mic sau egal decat timpul maxim de simulare se executa. Cand se termina aceasta bucla, programul se opreste. In urmatorul while se verifica daca mai avem clienti in asteptare si daca timpul curent este egal cu timpul de sosire a clientului. In Clasa ConcreteStrategyTime se alege in care coada ar fi cel mai avantajos sa se puna clientul cu timpul de sosire egal cu timpul curent, client care este de asemenea pe pozitia 0 in lista, deoarece am ordonat lista crescator in functie de timpul de sosire cu ajutorul clasei „SortByArrivalTime”, clasa care implementeaza interfata Comparator.Se verifica daca coada in care urmeaza sa fie pus clientul este goala, iar daca aceasta este goala se porneste un thread pe coada respectiva si se apeleaza metoda dispatchClient() care adauga clientul de pe pozitia 0 in coada cu timpul de procesare cel mai mic.Daca toate cozile au clienti, atunci doar se va adauga la coada cu cel mai mic timp de procesare.Dupa ce am adaugat clientul in coada il stergem din lista de asteptare.Bucla se reia daca mai avem clienti cu timpul de sosire egal cu timpul curent.Adormim thread-ul pentru o secunda pentru ca thread-urile din Clasa Queue sa execute o perioada de timp de simulare. Scriem in fisier la fiecare moment de timp starea listei clientilor in asteptare si starea cozilor. Dupa ce se iese din prima bucla, se inchide fisierul out si se afiseaza un mesaj in consola pentru a anunta ca programul si-a incheiat executia.
* Clasa Client- este clasa in care retinem detaliile pentru fiecare client in parte( id, timp de sosire, timp de procesare). Cu ajutorul metodei toString() afisam clientul in forma corespunzatoare cerintei temei.
* Clasa Scheduler- este clasa prin care alegem strategia de distributie in cozi si clasa in care adaugam clientii in cozi cu ajutorul metodei dispatchClient();
* Clasa ConcreteStrategyTime- este clasa care implementeaza interfata Strategy si a carei metode AddClient() returneaza indexul cozii cu cel mai mic timp de procesare, astfel incat sa adaugam clientul cat mai avantajos posibil.
* Clasa Queue- clasa care are ca si variabila instanta o coada de clienti si un AtomicInteger care retine timpul de procesare a cozii. Atunci cand este adaugat un client in coada, se adauga timpul de procesare a clientului la timpul de procesare a cozii. In metoda run() se proceseaza informatia din coada, iar cand coada se goleste, se opreste cu ajutorul unui return, deoarece se afla intr-o bucla infinita, while (true). Cat timp avem clienti in coada se ia clientul din varful cozii si se decrementeaza timpul de procesare (al cozii si al clientului). Cand acesta ajunge la 0, elementul din varf este scos din coada. Daca si timpul de procesare al cozii ajunge la 0, se iese. In caz contrar, se reia bucla cu urmatorul client aflat in varf.

4. Clase (metode si campuri importante)

* Clasa Manager – variabile instanta de tipul public int: numberOfClients (numarul de clienti), numberOfQueues (numarul de cozi), tMaxSim (timpul maxim de simulare), minArrivalTime (timpul minim de sosire), maxArrivalTime (timpul maxim de sosire), minWaitingTime (timpul minim de procesare/asteptare), maxWaitingTime (timpul maxim de procesare/asteptare). Variabila de tip FileWriter (fisierul in care se va scrie). Metode importante: metoda closed()- verifica daca mai sunt clienti in lista de asteptare sau in cozi. Metoda generateNRandomClients()- metoda care genereaza clienti a caror timp de sosire si de procesare sunt generate random in intervalul precizat in fisier. Totodata, metoda generateNRandomClients() ordoneaza crescator lista de clienti in functie de timpul de sosire. Metoda AverageWaiting() calculeaza timpul mediu de asteptare al tutror clientilor. Metoda PrintInFile() returneaza string-ul s (informatii despre lista de asteptare si cozi), string compus din apelul al altor doua functii PrintClients() care returneaza un string ce contine lista de clienti in asteptare si PrintQueues() care returneaza un string ce contine cozile. Tot in aceasta clasa se initializeaza lista de clienti si scheduler .
* In clasa Client avem ca variabile instanta :id, timpul de sosire si timpul de procesare.In afara de get-eri si set-eri mai avem metoda toString care ajuta la afisare.
* Clasa Queue- variabile instanta: coada de clienti de tipul BlockingQueue<Client>, AtomicInteger waitingPeriod (timpul de procesare), int number (numarul cozii). Pe langa set-eri, get-eri si metoda run() pe care am explicat-o la pasul anterior, mai avem metoda addClient() care adauga un client in coada .
* Clasa Scheduler- Aici avem Lista de cozi si obiectul de tip Strategy . Pe langa set-eri si get-eri avem metoda dispatchClient() care adauga clientul in coada cu cel mai mic timp de asteptare/procesare .

5.Rezultate

Avem 3 fisiere in, deci rezulta 3 fisiere out.

In primul fisier avem urmatoarele date de intrare: 4 clienti, 2 cozi, 60 timpul maxim de simulare, 2 timpul minim de sosire, 30 timpul maxim de sosire, 2 timpul minim de asteptare, 4 timpul maxim de asteptare.

Dupa ce am executat programul, avem astfel:

***Time 0***

***Waiting clients:(3,6,3);(4,14,2);(1,17,2);(2,17,3);***

***Queue1: closed***

***Queue2: closed***

S-au creat 4 clienti cu id-urile 1, 2, 3, 4. Primul client are id-ul 3, timpul de sosire 6 si timpul de procesare 3.Al doilea client are id-ul 4, timpul de sosire 14 si timpul de procesare 2. Al treilea client are id-ul 1, timpul de sosire 17 si timpul de procesare 2.Ultimul client are id-ul 2, timpul de sosire 17 si timpul de procesare 3.Se observa ca sunt inchise cozile 1 si 2. Acestea vor ramane inchise pana la timpul de simulare 6 cand primul client va ajunge.

***Time 6***

***Waiting clients:(4,14,2);(1,17,2);(2,17,3);***

***Queue1: (3,6,3)***

***Queue2: closed***

Se poate observa ca primul client a fost pus in prima coada si a disparut din lista de asteptare.

***Time 7***

***Waiting clients:(4,14,2);(1,17,2);(2,17,3);***

***Queue1: (3,6,2)***

***Queue2: closed***

La urmatorul moment de timp timpul de procesare a scazut cu 1. La momentul urmator, adica 8, ***Queue1:(3,6,1).***

***Time 9***

***Waiting clients:(4,14,2);(1,17,2);(2,17,3);***

***Queue1: closed***

***Queue2: closed***

La momentul 9 unde timpul de procesare ar trebui sa fie 0, clientul este eliminat din coada, iar coada este inchisa.

***Time 16***

***Waiting clients:(1,17,2);(2,17,3);***

***Queue1: closed***

***Queue2: closed***

***Time 17***

***Waiting clients:None***

***Queue1: (1,17,2)***

***Queue2: (2,17,3)***

***Time 18***

***Waiting clients:None***

***Queue1: (1,17,1)***

***Queue2: (2,17,2)***

***Time 19***

***Waiting clients:None***

***Queue1: closed***

***Queue2: (2,17,1)***

***Time 20***

***Waiting clients:None***

***Queue1: closed***

***Queue2: closed***

Se poate observa mai sus ca acelasi lucru se intampla si pentru al 3-lea si al 4-lea client. Deoarece acestia au timpul de sosire egal, iar ambele cozi sunt goale, clientii sunt pusi fiecare la cate o coada. La timpul 20 programul se opreste deoarece toti clientii din lista de asteptare au fost procesati, iar cozile s-au inchis. Daca timpul maxim de simulare ar fi fost mai mic de 20 programul s-ar fi terminat inainte ca toti clientii sa fie procesati.

In final avem: Average Time = 2.5 .

Cu aceste date de intrare nu avem foarte multa bataie de cap, deoarece in putine cazuri vor fi clienti care vor fi pusi la cozi care deja au clienti.

Al doilea fisier are ca si date de intrare:50 de clienti, 5 cozi, 60 timpul maxim de simulare, 2 timpul minim de sosire, 40 timpul maxim de sosire, 1 timpul minim de procesare si 7 timpul maxim de procesare.

***Time 0***

***Waiting clients:(1,2,4) ;(13,2,5);(29,2,6);(15,3,3)............(40,36,3);(42,37,4);(45,37,6);(28,39,5);***

***Queue1: closed***

***Queue2: closed***

***Queue3: closed***

***Queue4: closed***

***Queue5: closed***

La timpul 0 s-au generat cei 50 de clienti si toate cozile sunt goale.

***Time 34***

***Waiting clients:(6,35,4);(41,35,5);(40,36,3);(42,37,4);(45,37,6);(28,39,5);***

***Queue1: (7,32,2)***

***Queue2: (14,32,3)***

***Queue3: (36,34,5)***

***Queue4: (10,30,2)***

***Queue5: closed***

***Time 35***

***Waiting clients:(40,36,3);(42,37,4);(45,37,6);(28,39,5);***

***Queue1: (7,32,1)***

***Queue2: (14,32,2)***

***Queue3: (36,34,4)***

***Queue4: (10,30,1)***

***Queue5: (6,35,4)***

***Time 36***

***Waiting clients:(42,37,4);(45,37,6);(28,39,5);***

***Queue1: (41,35,5)***

***Queue2: (14,32,1)***

***Queue3: (36,34,3)***

***Queue4: (40,36,3)***

***Queue5: (6,35,3)***

La momentul 34 avem 2 clienti in asteptare cu timpul de sosire = 35 si o singura coada libera.La momentul 35 amandoi dispar din lista de asteptare, dar doar unul din ei apare intr-o coada. Celalalt nu a disparut, ci a fost pus intr-o coada cu timpul de procesare cel mai mic. Coada 1 are timpul de procesare 1, deci este pus in coada. La momentul urmator elementul din prima coada si-a terminat treaba si a lasat urmatorul client in fata.

Al treilea fisier are ca date de intrare:1000 de clienti, 20 de cozi, 200 timpul de simulare maxim, intre 10 si 100 timpul de sosire si intre 3 si 9 timpul de procesare. Acest exemplu este la fel ca cel precedent, doar ca la un nivel mult mai mare.

6.Concluzii

Aceasta tema m-a facut sa inteleg in sfarsit cum functioneaza thread-urile, sau cel putin mi-a facut o idee despre functionalitatea acestora. Lucrul pe care mi-a fost mai greu sa il inteleg este sincronizarea acestora si cum functioneaza metoda run().Cu siguranta programul poate suferi ajustari.Ceea ce am observat este ca atunci cand avem mai multi clienti si mai multe cozi, ocazional, se produc mici scapari.Cum ar fi: timpul de procesare este scazut cu 2 unitati si nu cu una sau nu este scazut deloc pentru o unitate de timp.Am intalnit destul de rar cazuri de genul, insa cred ca se poate imbunatati programul astfel incat sa nu mai aiba astfel de scapari.Am invatat cum se citeste si scrie intr-un fisier in java.

7.Bibligrafie

* <https://en.wikipedia.org/wiki/Java_concurrency>
* <http://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO11.pdf>
* http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/Assignment\_2/Java\_Concurrency.pdf